Assignment_1_Superpixels

October 12, 2021

1 Assignment 1: SLIC Superpixels.

Name: **Arunava Basu**

UID: **117720617**

Link to Google Drive :

https://colab.research.google.com/drive/17CCAi6JKpJSLH59zOElyxi6sXhOoCrN1?usp=sharing

Please submit a PDF containing all outputs to gradescope by **October 12, 5pm**

In this assignment, you will learn about superpixels. You will first generate superpixels by clustering pixels via k-means. This will generate a superpixel map such as the following:

You will make some adjustments to your k-means clustering. After reflection, you will then implement a better superpixel algorithm: SLIC, which lets you generate superpixel maps like the following:

1.1 Data

First, we download the MSRC labeled imaged database.

[ ]:
```
!wget http://download.microsoft.com/download/A/1/1/
 ↪A116CD80-5B79-407E-B5CE-3D5C6ED8B0D5/msrc_objcategimagedatabase_v1.zip
!unzip --qq msrc_objcategimagedatabase_v1.zip
# !pip3 install numpy --upgrade
```

```
--2021-10-12 17:44:19--  http://download.microsoft.com/download/A/1/1/A116CD80-5
B79-407E-B5CE-3D5C6ED8B0D5/msrc_objcategimagedatabase_v1.zip
Resolving download.microsoft.com (download.microsoft.com)… 72.246.252.126,
2600:1402:6800:283::e59, 2600:1402:6800:297::e59
Connecting to download.microsoft.com
(download.microsoft.com)|72.246.252.126|:80… connected.
HTTP request sent, awaiting response… 200 OK
```

```
Length: 44119839 (42M) [application/octet-stream]
Saving to: ‘msrc_objcategimagedatabase_v1.zip’

msrc_objcategimaged 100%[===================>]  42.08M   114MB/s    in 0.4s

2021-10-12 17:44:20 (114 MB/s) - ‘msrc_objcategimagedatabase_v1.zip’ saved
[44119839/44119839]
```

For this assignment, we will only use the following images. We define the list below as `im_list`.

[ ]:
```
im_list = ['MSRC_ObjCategImageDatabase_v1/1_22_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/1_27_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/3_3_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/3_6_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/6_5_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/7_19_s.bmp']
```

We provide the following functions as helpers for plotting your results. Please pay attention to their signatures and outputs.

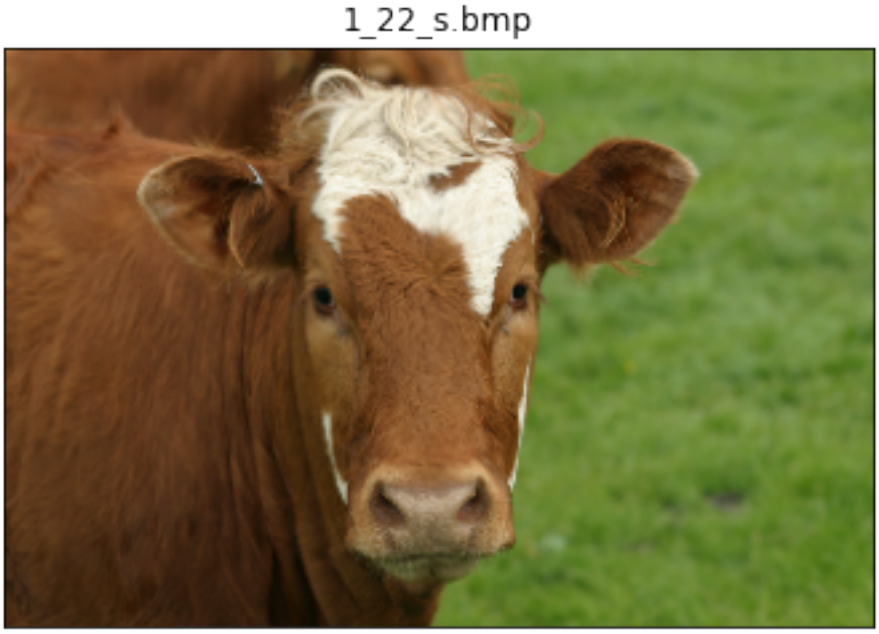
[ ]:
```
#All important functions to plot
%matplotlib inline
import cv2
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches

def plot_image(im,title,xticks=[],yticks= [],isCv2 = True):
    """
    im :Image to plot
    title : Title of image
    xticks : List of tick values. Defaults to nothing
    yticks :List of tick values. Defaults to nothing
    cv2 :Is the image cv2 image? cv2 images are BGR instead of RGB. Default True
    """
    plt.figure()
    if isCv2:
        im = im[:,:,::-1]
    plt.imshow(im)
    plt.title(title)
    plt.xticks(xticks)
    plt.yticks(yticks)

def superpixel_plot(im,seg,title = "Superpixels"):
    """
    Given an image (nXmX3) and pixelwise class mat (nXm),
    1. Consider each class as a superpixel
```

```
    2. Calculate mean superpixel value for each class
    3. Replace the RGB value of each pixel in a class with the mean value

    Inputs:
    im: Input image
    seg: Segmentation map
    title: Title of the plot

    Output: None
    Creates a plot
    """
    clust = np.unique(seg)
    mapper_dict = {i: im[seg == i].mean(axis = 0)/255. for i in clust}

    seg_img =  np.zeros((seg.shape[0],seg.shape[1],3))
    for i in clust:
        seg_img[seg == i] = mapper_dict[i]

    plot_image(seg_img,title)

    return

def rgb_segment(seg,n = None,plot = True,title=None,legend = True,color = None):
    """
    Given a segmentation map, get the plot of the classes
    """
    clust = np.unique(seg)
    if n is None:
        n = len(clust)
    if color is None:
        cm = plt.cm.get_cmap('hsv',n+1)
        # mapper_dict = {i:np.array(cm(i/n)) for i in clust}
        mapper_dict = {i:np.random.rand(3,) for i in clust}
    #elif color == 'mean':
        #TODO..get the mean color of cluster center and assign that to␣
↪mapper_dict

    seg_img =  np.zeros((seg.shape[0],seg.shape[1],3))
    for i in clust:
        seg_img[seg == i] = mapper_dict[i][:3]

    if plot:
        plot_image(seg_img,title = title)
    if legend:
        # get the colors of the values, according to the
        # colormap used by imshow
```

```
        patches = [ mpatches.Patch(color=mapper_dict[i], label=" : {l}".
↪format(l=i) ) for i in range(n) ]
        # put those patched as legend-handles into the legend
        plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc=2,␣
↪borderaxespad=0. )
        plt.grid(True)
        plt.show()

    return seg_img
```

For example, the following code uses `plot_image` to plot the 6 images we are using for this assignment.

[ ]:
```
for i in im_list:
    plot_image(cv2.imread(i),i.split("/")[-1])
```



1_27_s.bmp

3_3_s.bmp

3_6_s.bmp

6_5_s.bmp

1.2 Part 1: K-means on Image Pixels

For this portion of the assignment, you will attempt to identify superpixels by using k-means with various attributes of the image's pixels as inputs (r, g, b, and/or x, y).

1.2.1 Question 1: Perform k-means on image pixels (r, g, b). (20 points)

The k-means clustering algorithm is an unsupervised algorithm which, for some items and for some specified number of clusters represented by cluster centers, minimizes the distance between items and their associated cluster centers. It does so by iteratively assigning items to a cluster and recomputing the cluster center based on the assigned items.

Complete the pixel clustering function. It should take input an image (shape = `(n, m, 3)`) and number of clusters. Each pixel should be represented by a vector with 3 values: `(r, g, b)`.

Then, let our provided code plot the pixelwise and superpixel plots for the cow image (`1_22_s.bmp`), using your `cluster_pixels` implementation with the provided values for the number of clusters: `5, 10, 50`.

[ ]:
```
np.random.seed(828)
# np.set_printoptions(threshold=sys.maxsize)
from collections import OrderedDict
import sys
class Kmeans:

  def __init__(self, K=5, max_iters=10, colour_wt=1, post_wt=0):
```

```
    self.K=K
    self.max_iters=max_iters
    self.colour_wt=colour_wt
    self.post_wt=post_wt
    self.cluster={i:[] for i in range(self.K)}

    self.centroids=OrderedDict()

  def initialize_centroids(self):

    for i in range(self.K):

      point= (np.random.randint(0,self.image.shape[0]), np.random.
↪randint(0,self.image.shape[0]))

      pixel=self.image[point]
      self.centroids[i]=(pixel,point)


  def find_closest_centroid(self, centroids_list, current_point):

    # print(np.linalg.norm(centroids_list - current_point))

    current_point_pixel=current_point[0]
    current_point_pos=current_point[1]
    centroid_colours=[]
    centroid_pos=[]
    for i in self.centroids:
      centroid_colours.append(self.centroids[i][0])
      centroid_pos.append(self.centroids[i][1])

    dist_colour=np.linalg.norm(np.array(centroid_colours) - np.
↪array(current_point_pixel),axis=1)
    dist_pos=np.linalg.norm(np.array(centroid_pos)-np.
↪array(current_point_pos),axis=1)

    wtd_dist= self.colour_wt*dist_colour + self.post_wt*dist_pos

    return np.argmin(wtd_dist)


  def assign_clusters(self,seg_map):

    for i in range(self.image.shape[0]):
      for j in range(self.image.shape[1]):
        current_point = (self.image[i][j], (i,j))
```

```
        closest_cluster_id = self.find_closest_centroid(list(self.centroids.
↪values()), current_point)
        self.cluster[closest_cluster_id].append(current_point)
        seg_map[i][j] = closest_cluster_id

    return seg_map

  def get_new_centroids(self):
    cluster_pixels={i:[] for i in range(self.K)}
    cluster_pos={i:[] for i in range(self.K)}
    for i in self.cluster:

      for j in self.cluster[i]:

        cluster_pixels[i].append(j[0])
        cluster_pos[i].append(j[1])


    for i in self.centroids:
      self.centroids[i]=(np.mean(np.array(cluster_pixels[i]), axis=0), np.
↪mean(np.array(cluster_pos[i]), axis=0))


  def is_converged(self, old_centroids):

    centroid_colours=[]
    centroid_pos=[]
    old_centroid_colours=[]
    old_centroid_pos=[]

    for i in self.centroids:
      centroid_colours.append(self.centroids[i][0])
      centroid_pos.append(self.centroids[i][1])

    for i in old_centroids:

      old_centroid_colours.append(i[0])
      old_centroid_pos.append(i[1])
    print(np.array(centroid_colours).shape)
    print(np.array(old_centroid_colours).shape)
    diff_colour=np.array(centroid_colours) - np.array(old_centroid_colours)
    diff_pos=np.array(centroid_pos)-np.array(old_centroid_pos)

    dist_colour=np.sum(np.linalg.norm(diff_colour,axis=1))

    dist_pos=np.sum(np.linalg.norm(diff_pos,axis=1))
```

```
    wtd_dist= self.colour_wt*dist_colour + self.post_wt*dist_pos

    print(wtd_dist)
    return wtd_dist<5

  def fit(self, image):

    self.image=image

    self.initialize_centroids()

    seg_map=np.zeros(self.image.shape[:2])
    c=0

    for _ in range(self.max_iters):
      old_centroids=list(self.centroids.values())

      print("Iteration : ",c)
      seg_map=self.assign_clusters(seg_map)

      self.get_new_centroids()

      if self.is_converged(old_centroids):
        print("Converged ")
        break


      c+=1
    return seg_map
```

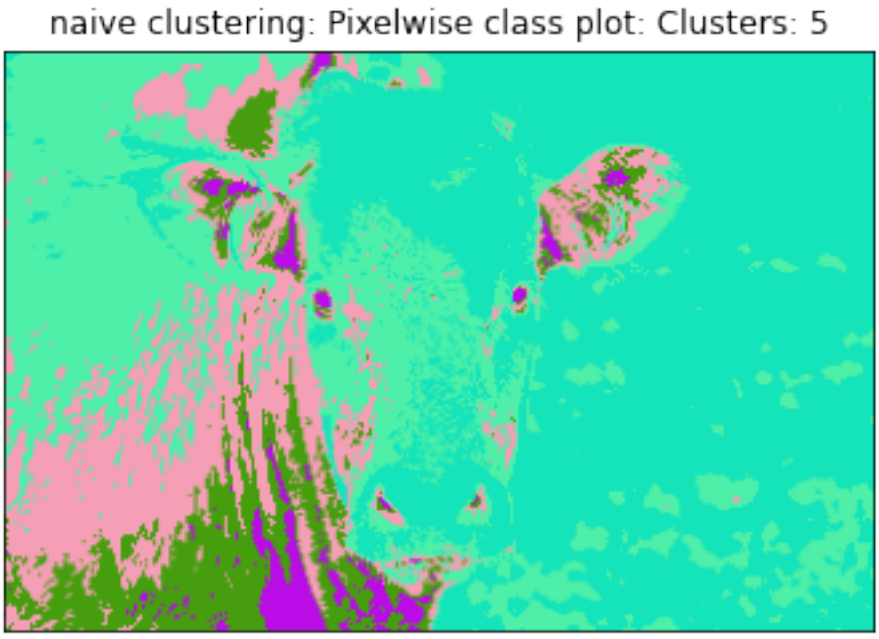
[ ]:
```
import numpy as  np


def cluster_pixels(im,k):
    #TODO Pixelwise clustering
    # assert 1==2," NOT IMPLEMENTED"
    #segmap is nXm. Each value in the 2D array is the cluster assigned to that␣
 ↪pixel
    k=Kmeans(K=k)

    return k.fit(im)

im=cv2.imread('MSRC_ObjCategImageDatabase_v1/1_22_s.bmp')
for k in [5,10,50]:
    clusters = cluster_pixels(im,k)
    _ = rgb_segment(clusters,n = k, title =  "naive clustering: Pixelwise class␣
 ↪plot: Clusters: " + str(k),legend = False)
```

```
    superpixel_plot(im,clusters,title =  "naive clustering: Superpixel plot:␣
↪Clusters: "+ str(k))
```

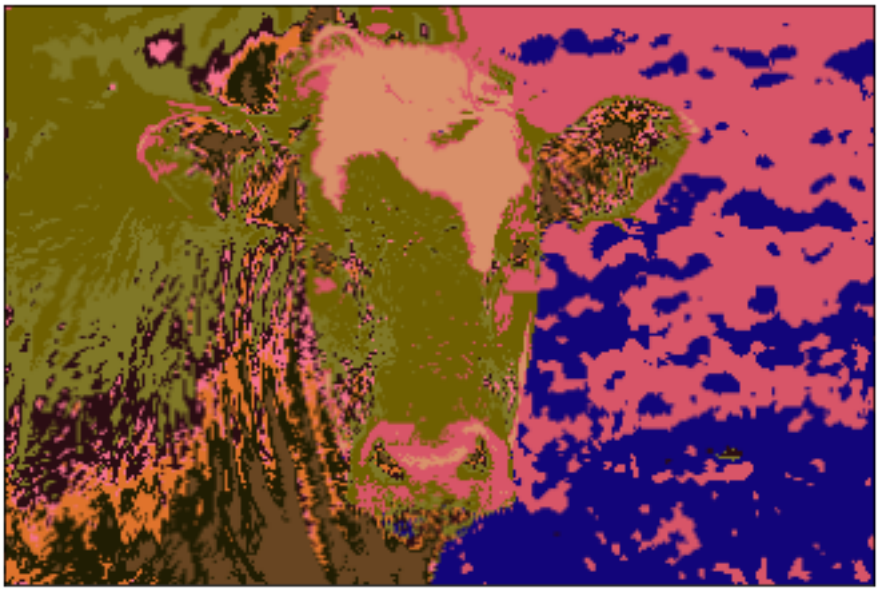
```
Iteration :  0
(5, 3)
(5, 3)
197.44773565504508
Iteration :  1
(5, 3)
(5, 3)
35.33160667754752
Iteration :  2
(5, 3)
(5, 3)
15.355311844047717
Iteration :  3
(5, 3)
(5, 3)
9.139516266950581
Iteration :  4
(5, 3)
(5, 3)
6.398940412288776
Iteration :  5
(5, 3)
(5, 3)
4.902260832264809
Converged
Iteration :  0
(10, 3)
(10, 3)
658.8336856802224
Iteration :  1
(10, 3)
(10, 3)
45.911259044034416
Iteration :  2
(10, 3)
(10, 3)
18.927299110084498
Iteration :  3
(10, 3)
(10, 3)
11.046362717833464
Iteration :  4
(10, 3)
(10, 3)
```

```
7.72467310122476
Iteration :  5
(10, 3)
(10, 3)
5.9111925952633895
Iteration :  6
(10, 3)
(10, 3)
4.682644671560087
Converged
Iteration :  0
(50, 3)
(50, 3)
817.013217439205
Iteration :  1
(50, 3)
(50, 3)
98.81559514807462
Iteration :  2
(50, 3)
(50, 3)
42.059036812159015
Iteration :  3
(50, 3)
(50, 3)
26.971968038873733
Iteration :  4
(50, 3)
(50, 3)
19.100184306985064
Iteration :  5
(50, 3)
(50, 3)
14.62286998544273
Iteration :  6
(50, 3)
(50, 3)
11.675439116447135
Iteration :  7
(50, 3)
(50, 3)
9.661058983227408
Iteration :  8
(50, 3)
(50, 3)
8.221335596471837
Iteration :  9
(50, 3)
```

```
(50, 3)
7.01028070746151
```

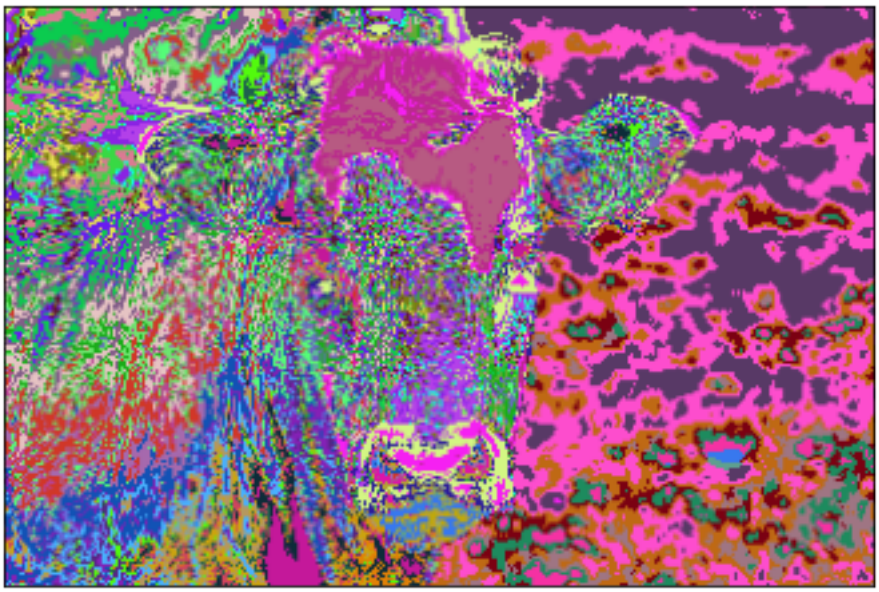


naive clustering: Pixelwise class plot: Clusters: 10



naive clustering: Superpixel plot: Clusters: 10

naive clustering: Pixelwise class plot: Clusters: 50



naive clustering: Superpixel plot: Clusters: 50

1.2.2 Question 2: Perform k-means on image pixels (`r, g, b, x, y`). (10 points)

Instead of the clustering running on (r,g,b) values, run the clustering on (r,b,g,x,y).

Try with clusters = `5,10,25,50,150`. You only need to plot the cow (`1_22_s.bmp`). _____

[ ]:
```
#TODO: clustering r,b,g,x,y values

def cluster_rgbxy(im,k):
    """
    Given image im and asked for k clusters, return nXm size 2D array
    segmap[0,0] is the class of pixel im[0,0,:]
    """
    # assert 1==2,"NOT IMPLEMENTED
    k=Kmeans(K=k, post_wt=1)

    return k.fit(im)

im=cv2.imread('MSRC_ObjCategImageDatabase_v1/1_22_s.bmp')
for k in [5, 10, 25, 50, 150]:
    clusters = cluster_rgbxy(im,k)
    _ = rgb_segment(clusters,n = k, title =  "naive clustering: Pixelwise class␣
 ↪plot: Clusters: " + str(k),legend = False)
    superpixel_plot(im,clusters,title =  "naive clustering: Superpixel plot:␣
 ↪Clusters: "+ str(k))
```

```
Iteration :  0
(5, 3)
(5, 3)
366.2731790550635
Iteration :  1
(5, 3)
(5, 3)
104.9795423865246
Iteration :  2
(5, 3)
(5, 3)
45.11251310564778
Iteration :  3
(5, 3)
(5, 3)
26.824822051436854
Iteration :  4
(5, 3)
(5, 3)
18.62554604244678
Iteration :  5
(5, 3)
(5, 3)
15.072497648104429
Iteration :  6
(5, 3)
(5, 3)
```

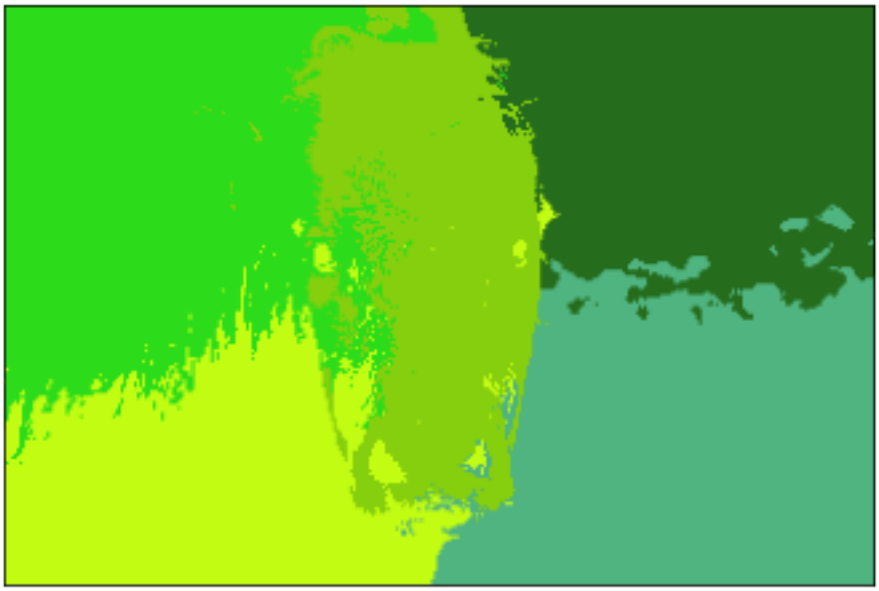
```
13.470578997111442
Iteration :  7
(5, 3)
(5, 3)
12.255710513736204
Iteration :  8
(5, 3)
(5, 3)
11.191847134689873
Iteration :  9
(5, 3)
(5, 3)
9.925127652193314
Iteration :  0
(10, 3)
(10, 3)
794.0352795887593
Iteration :  1
(10, 3)
(10, 3)
200.45634758345352
Iteration :  2
(10, 3)
(10, 3)
90.43273490123552
Iteration :  3
(10, 3)
(10, 3)
52.72219993313452
Iteration :  4
(10, 3)
(10, 3)
35.207053951877974
Iteration :  5
(10, 3)
(10, 3)
25.861363389100035
Iteration :  6
(10, 3)
(10, 3)
20.467476402610302
Iteration :  7
(10, 3)
(10, 3)
17.181781239723193
Iteration :  8
(10, 3)
(10, 3)
```

```
14.95567069912363
Iteration :  9
(10, 3)
(10, 3)
13.426356905724944
Iteration :  0
(25, 3)
(25, 3)
1525.7756383712508
Iteration :  1
(25, 3)
(25, 3)
278.0379545046613
Iteration :  2
(25, 3)
(25, 3)
108.6074500474899
Iteration :  3
(25, 3)
(25, 3)
67.23166134593171
Iteration :  4
(25, 3)
(25, 3)
47.41573284420617
Iteration :  5
(25, 3)
(25, 3)
36.65086188496508
Iteration :  6
(25, 3)
(25, 3)
29.794388966656733
Iteration :  7
(25, 3)
(25, 3)
24.83392897238994
Iteration :  8
(25, 3)
(25, 3)
20.974296129351252
Iteration :  9
(25, 3)
(25, 3)
18.124756519514
Iteration :  0
(50, 3)
(50, 3)
```

```
2973.7552681650022
Iteration :  1
(50, 3)
(50, 3)
382.2634142538826
Iteration :  2
(50, 3)
(50, 3)
170.4123096666375
Iteration :  3
(50, 3)
(50, 3)
103.12084298990914
Iteration :  4
(50, 3)
(50, 3)
72.99792799496973
Iteration :  5
(50, 3)
(50, 3)
55.94349133996752
Iteration :  6
(50, 3)
(50, 3)
45.05359072239156
Iteration :  7
(50, 3)
(50, 3)
37.67870787713004
Iteration :  8
(50, 3)
(50, 3)
32.88436640870691
Iteration :  9
(50, 3)
(50, 3)
29.000857089650417
Iteration :  0
(150, 3)
(150, 3)
5080.007873613738
Iteration :  1
(150, 3)
(150, 3)
977.2377797216781
Iteration :  2
(150, 3)
(150, 3)
```

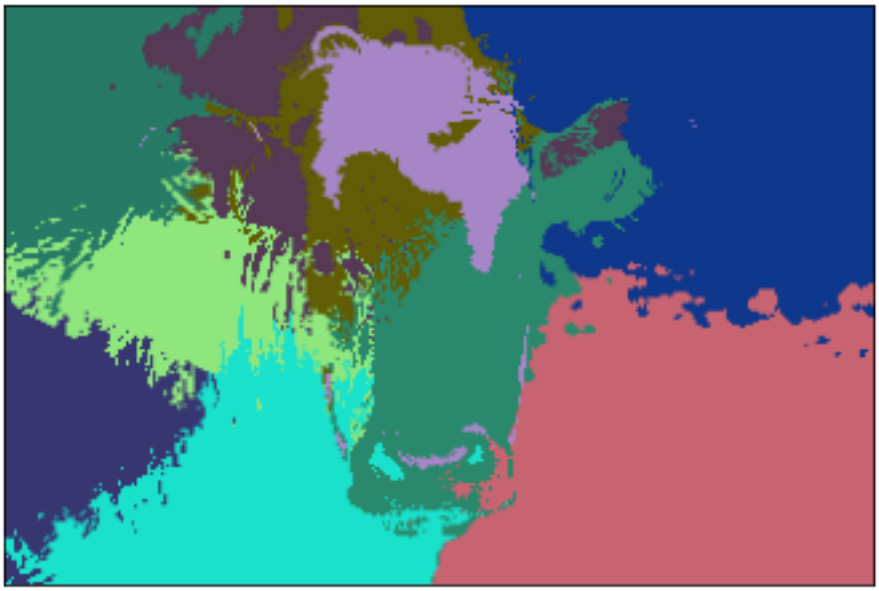
```
400.3793545865668
Iteration :  3
(150, 3)
(150, 3)
232.01847075064057
Iteration :  4
(150, 3)
(150, 3)
157.82415864753753
Iteration :  5
(150, 3)
(150, 3)
118.69462542539421
Iteration :  6
(150, 3)
(150, 3)
95.40363769726963
Iteration :  7
(150, 3)
(150, 3)
78.9668947190567
Iteration :  8
(150, 3)
(150, 3)
67.36045158559297
Iteration :  9
(150, 3)
(150, 3)
58.286600290289556
```

naive clustering: Pixelwise class plot: Clusters: 5



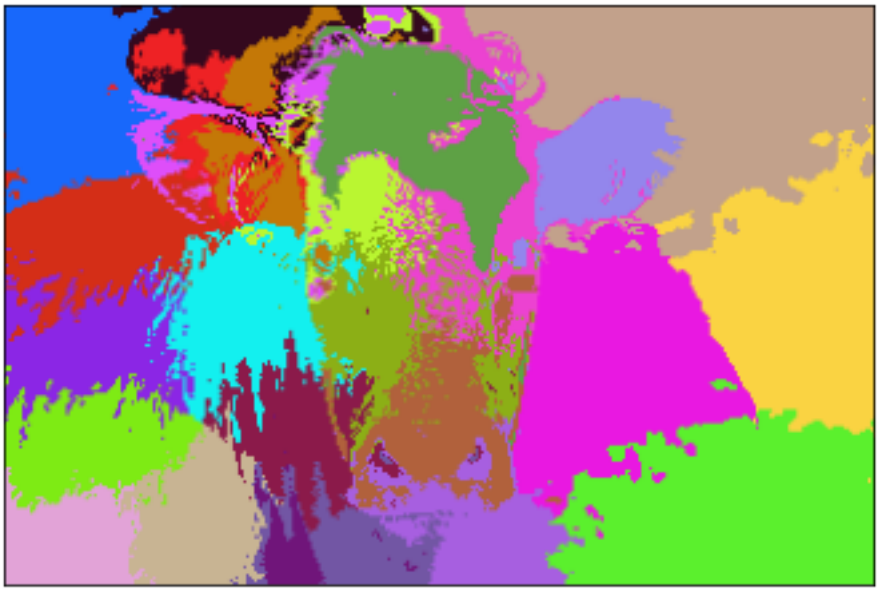
naive clustering: Superpixel plot: Clusters: 5

naive clustering: Pixelwise class plot: Clusters: 10



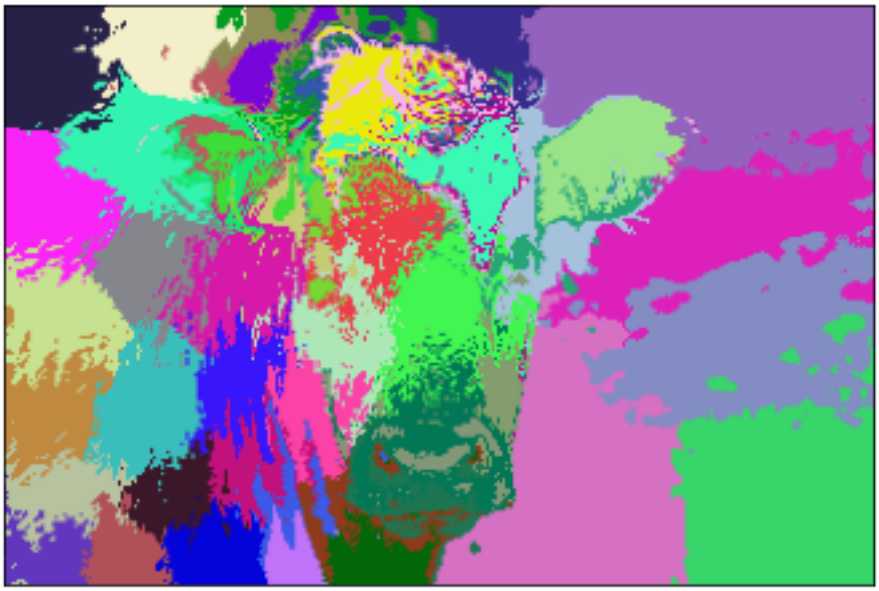
naive clustering: Superpixel plot: Clusters: 10

naive clustering: Pixelwise class plot: Clusters: 25



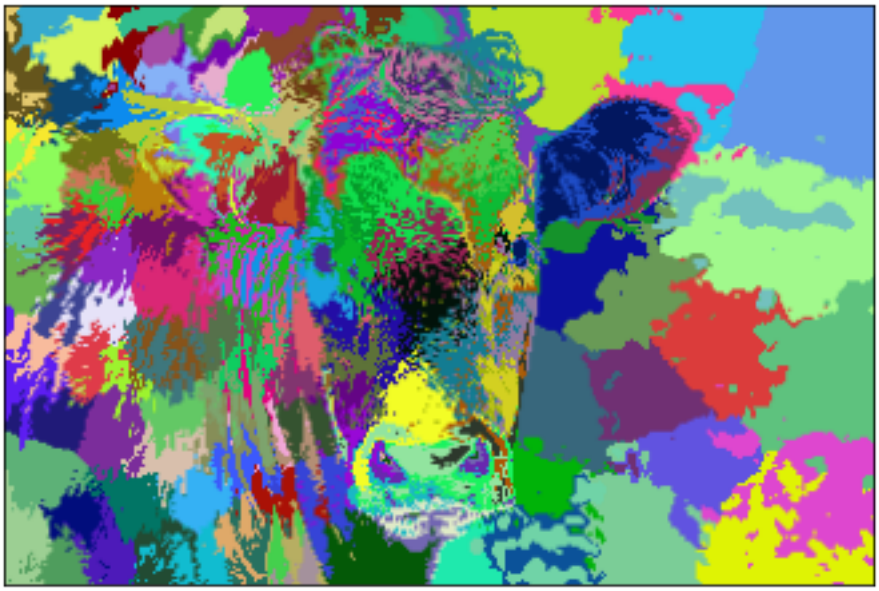
naive clustering: Superpixel plot: Clusters: 25

naive clustering: Pixelwise class plot: Clusters: 50



naive clustering: Superpixel plot: Clusters: 50

naive clustering: Pixelwise class plot: Clusters: 150



naive clustering: Superpixel plot: Clusters: 150

1.2.3 Question 3: What advantage did the (r, g, b, x, y) clustering give, compared to (r, g, b)? Please answer in 3 sentences or fewer. (5 points)

Your Answer: RGBXY clustering takes into account the spatial relations between pixels as opposed to RGB clustering which considers only the colour space in the relational mapping. Intuitively, we know that similar kinds of pixels tend to be spatially closer to each other, therefore, including the position gives us a more rich feature space to cluster in. Also, visually more consistent superpixels are formed with this method.

1.2.4 Question 4: K-means on image pixels (r, g, b, x, y) with weighted distances. (15 points)

Let: $cluster\_center_i$ be i^{th} cluster center. $cluster\_center_i^{rgb}$ be the rgb value and $cluster\_center_i^{xy}$ be the corresponding coordinate of the pixel.

Let x_{rgb} be the the rgb value of a point Let x_{xy} be the coordinate of x_{rgb}

Use the following distance function for k-means: $distance(x_{rgb}, x_{xy}) = \lambda_1 * euclidean(x_{rgb}, cluster\_center_i^{rgb}) + \lambda_2 * euclidean(x_{xy}, cluster\_center_i^{xy})$

Find good values for λ_1 and λ_2 for 250 clusters.

With those good values, **plot results for all 6 images in im_list.**

[ ]:
```
#TODO: clustering r,b,g,x,y values with lambdas
def cluster_rgbxy_with_weighted_dist(im,k, l1, l2):
    """
    Given image im and asked for k clusters, return nXm size 2D array
    segmap[0,0] is the class of pixel im[0,0,:]
    """
    #assert 1==2,"NOT IMPLEMENTED"
    k=Kmeans(K=k,colour_wt=l1, post_wt=l2)
    return k.fit(im)


for i in im_list:
  im = cv2.imread(i)
  k = 250
  clusters = cluster_rgbxy_with_weighted_dist(im, k, l1=20, l2=10)
  _ = rgb_segment(clusters,n = k, title =  "weighted clustering with RGBXY␣
↪vector: Pixelwise class plot: Clusters: " + str(k),legend = False)
  superpixel_plot(im,clusters,title =  "weighted clustering with RGBXY vector:␣
↪Superpixel plot: Clusters: "+ str(k))
```

```
Iteration :  0
(250, 3)
(250, 3)
89948.00615251582
Iteration :  1
(250, 3)
```

```
(250, 3)
15174.231624831114
Iteration :  2
(250, 3)
(250, 3)
7249.042014318149
Iteration :  3
(250, 3)
(250, 3)
4648.090289800333
Iteration :  4
(250, 3)
(250, 3)
3353.467586136058
Iteration :  5
(250, 3)
(250, 3)
2541.3089427418963
Iteration :  6
(250, 3)
(250, 3)
2035.2451740080955
Iteration :  7
(250, 3)
(250, 3)
1674.9145736535472
Iteration :  8
(250, 3)
(250, 3)
1419.4390274879681
Iteration :  9
(250, 3)
(250, 3)
1227.913928970841
Iteration :  0
(250, 3)
(250, 3)
111871.55030809883
Iteration :  1
(250, 3)
(250, 3)
16968.54603715671
Iteration :  2
(250, 3)
(250, 3)
8000.356132406985
Iteration :  3
(250, 3)
```

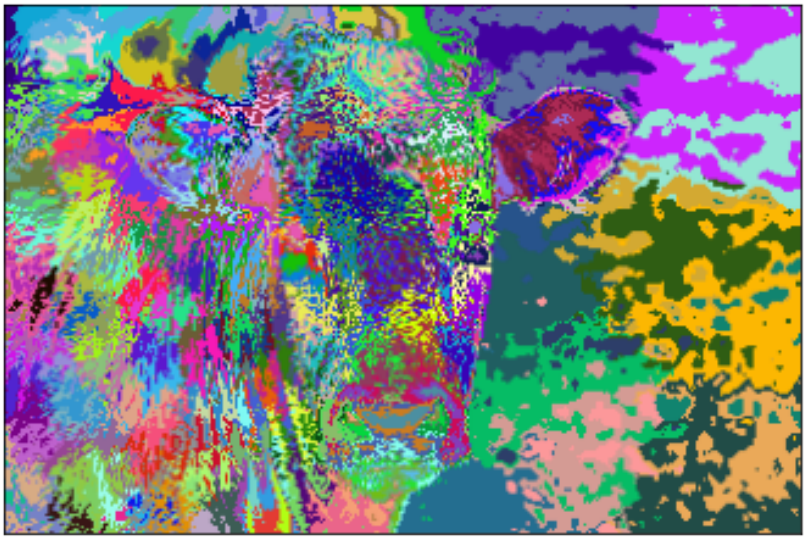
```
(250, 3)
5050.837554734047
Iteration :  4
(250, 3)
(250, 3)
3534.6504814468108
Iteration :  5
(250, 3)
(250, 3)
2666.7206454500765
Iteration :  6
(250, 3)
(250, 3)
2133.514422186909
Iteration :  7
(250, 3)
(250, 3)
1781.20721846837
Iteration :  8
(250, 3)
(250, 3)
1520.9554653007685
Iteration :  9
(250, 3)
(250, 3)
1316.7380008191622
Iteration :  0
(250, 3)
(250, 3)
121761.4020630406
Iteration :  1
(250, 3)
(250, 3)
17574.496650456982
Iteration :  2
(250, 3)
(250, 3)
8761.530530941316
Iteration :  3
(250, 3)
(250, 3)
5528.8127929358425
Iteration :  4
(250, 3)
(250, 3)
3888.6062132274074
Iteration :  5
(250, 3)
```

```
(250, 3)
2950.9210928632765
Iteration :  6
(250, 3)
(250, 3)
2377.456968647698
Iteration :  7
(250, 3)
(250, 3)
1961.322476216224
Iteration :  8
(250, 3)
(250, 3)
1655.0882549063917
Iteration :  9
(250, 3)
(250, 3)
1425.759746594213
Iteration :  0
(250, 3)
(250, 3)
104474.09293842912
Iteration :  1
(250, 3)
(250, 3)
16013.026656789876
Iteration :  2
(250, 3)
(250, 3)
7723.52388834368
Iteration :  3
(250, 3)
(250, 3)
4962.568522776721
Iteration :  4
(250, 3)
(250, 3)
3643.037711876094
Iteration :  5
(250, 3)
(250, 3)
2843.6274854358585
Iteration :  6
(250, 3)
(250, 3)
2321.532895833924
Iteration :  7
(250, 3)
```

```
(250, 3)
1962.6357216217316
Iteration :  8
(250, 3)
(250, 3)
1689.108400610869
Iteration :  9
(250, 3)
(250, 3)
1463.3260341629252
Iteration :  0
(250, 3)
(250, 3)
106575.80790693389
Iteration :  1
(250, 3)
(250, 3)
18901.151367992203
Iteration :  2
(250, 3)
(250, 3)
8317.531604781136
Iteration :  3
(250, 3)
(250, 3)
5175.34661545929
Iteration :  4
(250, 3)
(250, 3)
3776.920428241814
Iteration :  5
(250, 3)
(250, 3)
2866.84762570174
Iteration :  6
(250, 3)
(250, 3)
2347.0303804608434
Iteration :  7
(250, 3)
(250, 3)
1978.188146121433
Iteration :  8
(250, 3)
(250, 3)
1685.438289204358
Iteration :  9
(250, 3)
```

```
(250, 3)
1455.1728771753199
Iteration :  0
(250, 3)
(250, 3)
214973.64097774873
Iteration :  1
(250, 3)
(250, 3)
31951.55866455967
Iteration :  2
(250, 3)
(250, 3)
13832.11879107944
Iteration :  3
(250, 3)
(250, 3)
8804.52076128708
Iteration :  4
(250, 3)
(250, 3)
6304.3403050214665
Iteration :  5
(250, 3)
(250, 3)
4902.079191336708
Iteration :  6
(250, 3)
(250, 3)
4030.8673365897976
Iteration :  7
(250, 3)
(250, 3)
3391.2152878378456
Iteration :  8
(250, 3)
(250, 3)
2931.2858789968122
Iteration :  9
(250, 3)
(250, 3)
2532.3472353619036
```

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250

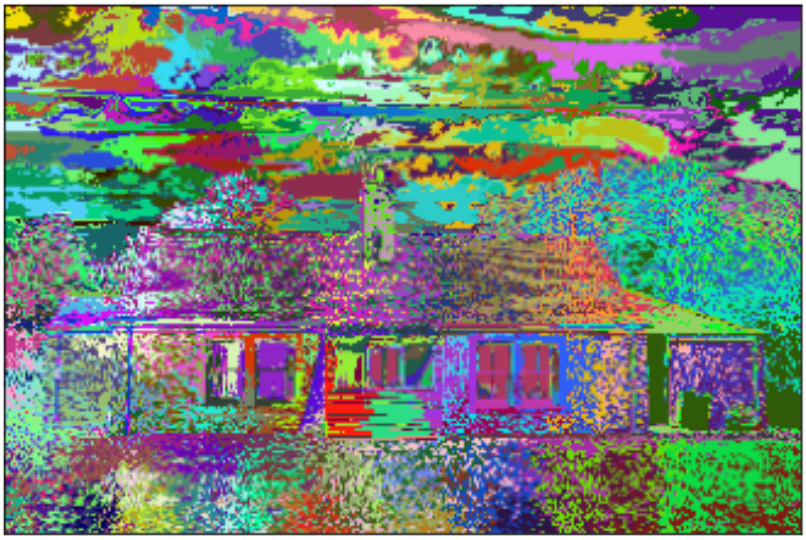


weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250

weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250



weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250

weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250

weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

weighted clustering with RGBXY vector: Pixelwise class plot: Clusters: 250

weighted clustering with RGBXY vector: Superpixel plot: Clusters: 250

1.2.5 Question 5: Replicate SLIC (50 points)

It doesn't look like we have a very favourable outcome with superpixels simply being implemented. Can we do better? Have a look at the SLIC paper here. Incorporate S and m and redefine your distance metric as per the paper.

Finding an existing implementation of SLIC and using it for your assigment would be

considered cheating.

[ ]:
```
import math

def rgb_to_lab(img):
    img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB).astype(np.float64)
    return img_lab

def get_superPixel(h, w, img):

        return SuperPixel(h, w, img[h,w][0], img[h,w][1], img[h,w][2])

def initialize_centers(S,img, img_height, img_width, clusters):

        h= S // 2
        w= S // 2

        while h < img_height:
                while w < img_width:
                        clusters.append(get_superPixel(h, w, img))
                        w=w+S
                w= S // 2
                h= h + S

        return clusters

def compute_gradient(h, w, img, img_width, img_height):



        a=img[h+1, w] - img[h-1,w]
        b=img[h, w+1] - img[h, w-1]

        grad=np.linalg.norm(a)**2 + np.linalg.norm(b)**2

        return grad



def assign_cluster_center_to_min_grad(clusters, img):

        for i in clusters:
                cluster_grad= compute_gradient(i.h, i.w, img, img_width,␣
 ↪img_height)

                for j in range(-1,2):
                        for k in range(-1,2):
```

```
                                new_h = i.h + j
                                new_w = i.w + k

                                new_grad=compute_gradient(new_h, new_w, img,␣
↪img_width, img_height)

                                if new_grad< cluster_grad:

                                        i.update(new_h, new_w, img[new_h,␣
↪new_w][0], img[new_h, new_w][1], img[new_h, new_w][2])
                                        cluster_grad=new_grad

def assign_pixels_to_clusters(clusters, S, img, img_height, img_width, flag,␣
↪distance, seg_map):

        for idx, i in enumerate(clusters):
                for j in range(i.h-2*S, i.h + 2*S):
                        if j<0 or j>=img_height:
                                continue

                        for k in range(i.w -2*S, i.w + 2*S):
                                if k<0 or k>= img_width:
                                        continue

                                l, a, b = img[j, k]
                                Dc= math.sqrt(math.pow(l - i.l,2)+math.pow(a-i.
↪a,2)+math.pow(b-i.b,2))

                                Ds=math.sqrt(math.pow(j-i.h,2)+math.pow(k-i.
↪w,2))
                                D=Dc+(m/S)*Ds

                                if D < distance[j,k]:
                                        if (j, k) not in flag:
                                                flag[(j, k)] = i
                                                i.pixels.append((j,k))
                                        else:
                                                flag[(j,k)].pixels.remove((j,k))
                                                flag[(j,k)]=i
                                                i.pixels.append((j,k))

                                        distance[j,k]=D
                                        seg_map[j,k]=idx

        return seg_map
```

```
def get_new_centers(clusters):

        for i in clusters:

                s_h=0
                s_w=0
                n=0

                for j in i.pixels:
                        s_h=s_h+ j[0]
                        s_w=s_w+ j[1]
                        n+=1

                s_h=s_h//n
                s_w=s_w//n
                i.update(s_h, s_w, img[s_h,s_w][0], img[s_h,s_w][1], img[s_h,␣
 ↪s_w][2])

def slic_segmentation(S, img, img_height, img_width, clusters, flag, distance):

        clusters=initialize_centers(S, img, img_height, img_width, clusters)
        assign_cluster_center_to_min_grad(clusters,img)
        seg_map=np.zeros(img.shape[:2])
        for i in range(10):
                print("Iteration : ", i)
                seg_map=assign_pixels_to_clusters(clusters, S, img, img_height,␣
 ↪img_width, flag, distance, seg_map)

                get_new_centers(clusters)


        return clusters, seg_map



class SuperPixel(object):

        def __init__(self, h, w, l=0, a=0, b=0):

                self.update(h, w, l, a, b)
                self.pixels=[]

        def update(self, h, w, l, a, b):

                self.h=h
                self.w=w
```

```
                self.l=l
                self.a=a
                self.b=b
```

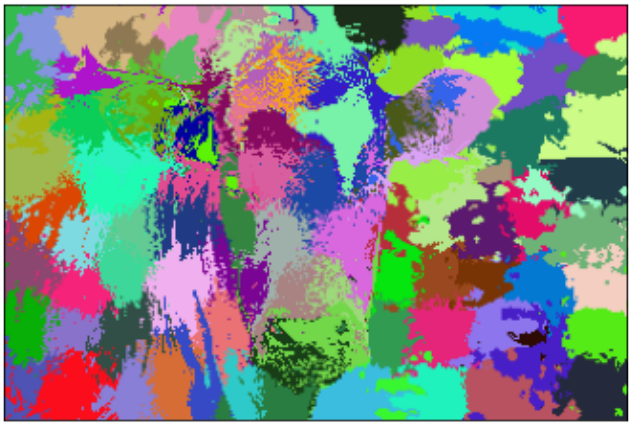
[ ]:
```
############Algorithm#############
#Compute grid steps: S
#you can explore different values of m
#initialize cluster centers [l,a,b,x,y] using
#Perturb for minimum G
#while not converged
##for every pixel:
####  compare distance D_s with each cluster center within 2S X 2S.
####  Assign to nearest cluster
##calculate new cluster center


K=100
m=10



def SLIC(im, k):
    """
    Input arguments:
    im: image input
    k: number of cluster segments


    Compute
    S: As described in the paper
    m: As described in the paper (use the same value as in the paper)
    follow the algorithm..

    returns:
    segmap: 2D matrix where each value corresponds to the image pixel's cluster␣
 ↪number
    """


    N=img_width*img_height
    S=int(math.sqrt(N/K))

    clusters=[]

    flag={}

    distance=np.full((img_height,img_width), np.inf)
```

```
    _, seg_map=slic_segmentation(S, img, img_height, img_width, clusters, flag,␣
↪distance)

    return seg_map




for i in im_list:
  img_rgb=cv2.imread(i)
  img_height=img_rgb.shape[0]
  img_width=img_rgb.shape[1]
  img= rgb_to_lab(img_rgb)
  clusters = SLIC(img, K)
  _ = rgb_segment(clusters,n = K, title =  "SLIC superpixel clustering without␣
↪connectivity enforcement: Pixelwise class plot: Clusters: " + str(K),legend␣
↪= False)
  superpixel_plot(img_rgb,clusters,title =  "SLIC superpixel clustering without␣
↪connectivity enforcement: Superpixel plot: Clusters: "+ str(K))
```

```
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
```

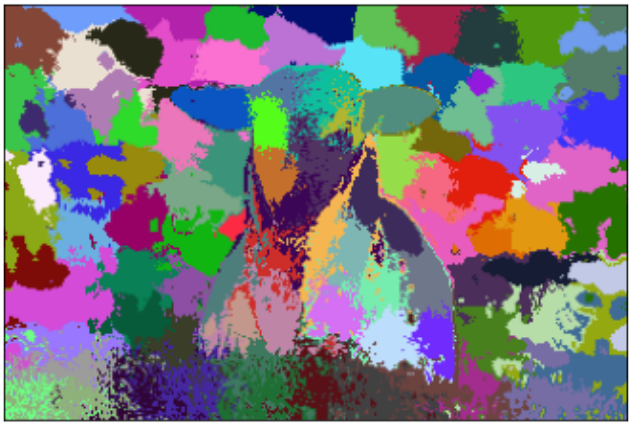
```
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
```

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



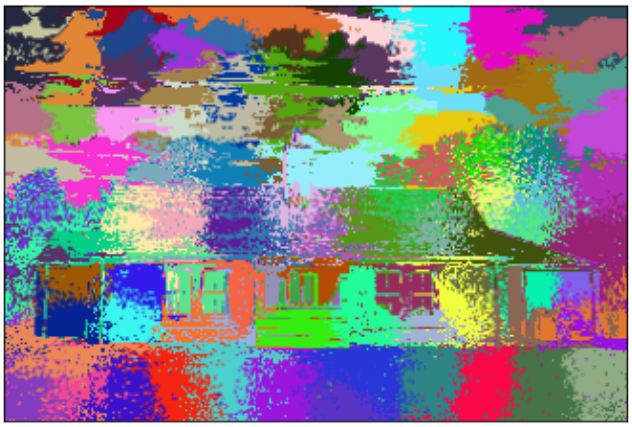
SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

With SLIC implemented, plot results for all 6 images.

[ ]:
```
## TODO: Call our plot functions with your SLIC results for all 6 images
```

1.2.6 Bonus Question: Enforce connectivity (20 points)

There are many superpixels which are very small and disconnected from each other. Merge them with larger superpixels

O(N) algorithm: 1. Set minimum size of superpixel 2. If region smaller than threshold, assign to nearest cluster

Plot results for the 6 images.

[ ]:
```
def find_nearest_cluster_id(i,clusters, N,S):

        l=i.l
        a=i.a
        b=i.b
        h=i.h
        k=i.w
        min_dist=0
        min_dist_id=0
        for n, j in enumerate(clusters):

                if(N==n):
                        continue
```

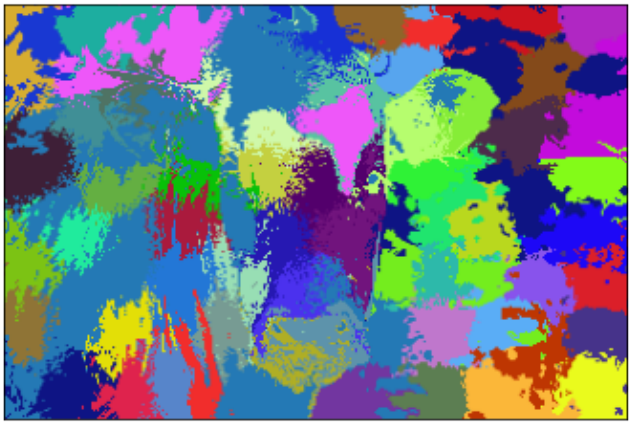
```
                Dc= math.sqrt(math.pow(l - j.l,2)+math.pow(a-j.a,2)+math.
↪pow(b-j.b,2))
                Ds=math.sqrt(math.pow(h-j.h,2)+math.pow(k-j.w,2))
                D=Dc+(m/S)*Ds

                if min_dist<D:
                        min_dist=D
                        min_dist_id=n
        # print(min_dist_id)
        return min_dist_id


def SLIC_enforce_connectivity(im, k):
    """
    Input arguments:
    im: image input
    k: number of cluster segments


    Compute
    S: As described in the paper
    m: As described in the paper (use the same value as in the paper)
    follow the algorithm..

    returns:
    segmap: 2D matrix where each value corresponds to the image pixel's cluster␣
↪number
    **enforce connectivity has been implemented
    """


    N=img_width*img_height
    S=int(math.sqrt(N/K))

    clusters=[]

    flag={}

    distance=np.full((img_height,img_width), np.inf)

    min_area=img_height*img_width/K
    clusters, seg_map=slic_segmentation(S, img, img_height, img_width,␣
↪clusters, flag, distance)
    unique,counts=np.unique(seg_map, return_counts=True)
    cluster_counts=dict(zip(unique.astype(int),counts))

    new_seg_map=seg_map
```

```
    for n, i in enumerate(clusters):

      curr_cluster=cluster_counts[n]
      if (curr_cluster<min_area):

        new_cluster_id= find_nearest_cluster_id(i,clusters,n,S)
        new_seg_map=np.where(new_seg_map==n,new_cluster_id,new_seg_map)

    return new_seg_map


K=100
m=10



for i in im_list:
  img_rgb=cv2.imread(i)
  img_height=img_rgb.shape[0]
  img_width=img_rgb.shape[1]
  img= rgb_to_lab(img_rgb)
  clusters = SLIC_enforce_connectivity(img, K)
  _ = rgb_segment(clusters,n = K, title =  "SLIC superpixel clustering without␣
 ↪connectivity enforcement: Pixelwise class plot: Clusters: " + str(K),legend␣
 ↪= False)
  superpixel_plot(img_rgb,clusters,title =  "SLIC superpixel clustering without␣
 ↪connectivity enforcement: Superpixel plot: Clusters: "+ str(K))
```

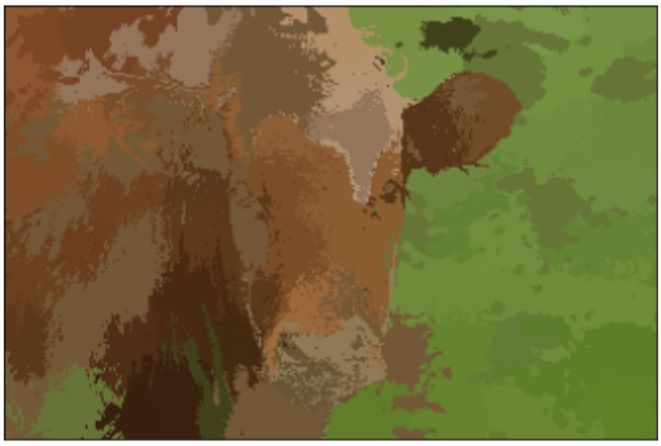
```
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
```

```
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
Iteration :  0
Iteration :  1
Iteration :  2
Iteration :  3
Iteration :  4
Iteration :  5
Iteration :  6
Iteration :  7
Iteration :  8
Iteration :  9
```

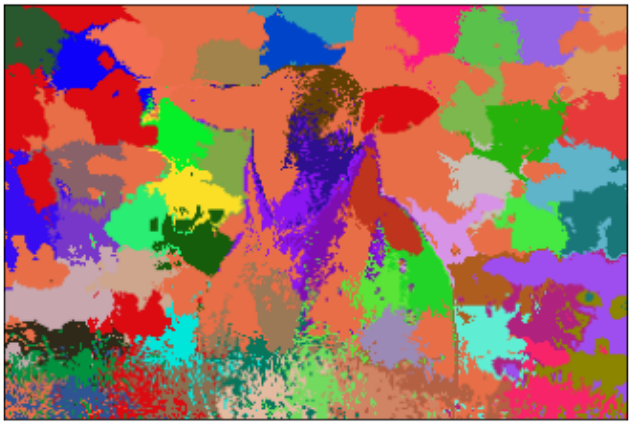
SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

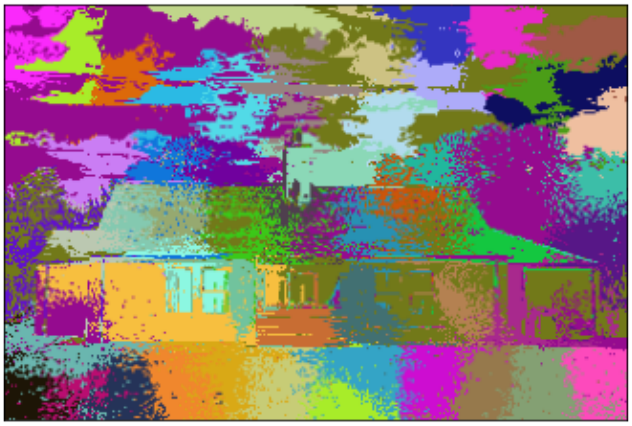


SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100



SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Pixelwise class plot: Clusters: 100

SLIC superpixel clustering without connectivity enforcement: Superpixel plot: Clusters: 100